homework

2026 年 1 月 17 日

0.1 基于深度卷积神经网络——结直肠癌——病理图像分类

0.1.1 一、数据集介绍

采用 **NCT-CRC-HE-100K** 公开数据集：* **数据规模**：包含 100,000 张互不重叠的病理图像块 (Patches)。* **图像规格**：224x224 像素，分辨率为 0.5 µm/pixel (MPP)，已进行 Macenko 颜色归一化。* **组织类别**：涵盖 9 种常见的结直肠癌组织类型，包括：脂肪 (ADI)、背景 (BACK)、碎片 (DEB)、淋巴细胞 (LYM)、粘液 (MUC)、平滑肌 (MUS)、正常粘膜 (NORM)、癌相关基质 (STR) 和结直肠腺癌上皮 (TUM)。

0.1.2 二、技术路线

本作业基于 **PyTorch** 框架，采用 **ResNet50** 进行迁移学习。主要流程包括：数据预处理与增强、探索性数据分析、模型训练、以及基于 Grad-CAM 的可解释性分析。

0.1.3 Step 1 数据读取

基础配置

[15]:
```
import os
import sys
import random
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, models, transforms
from torch.utils.data import DataLoader, random_split
```

```
from sklearn.metrics import confusion_matrix, classification_report, roc_curve,␣
 ↪auc
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.preprocessing import label_binarize
from itertools import cycle
import matplotlib.pyplot as plt
from PIL import Image
import seaborn as sns
from pathlib import Path
from tqdm import tqdm
import zipfile




# 确保可复现性，固定随机种子
def set_seed(seed=2026):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True

set_seed()

# 选择 GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

数据解压与读取

[2]:
```
data_dir = Path("./NCT-CRC-HE-100K")
```

[3]:
```
# 解压数据集
zip_path = "archive.zip"

with zipfile.ZipFile(zip_path, 'r') as zf:
```

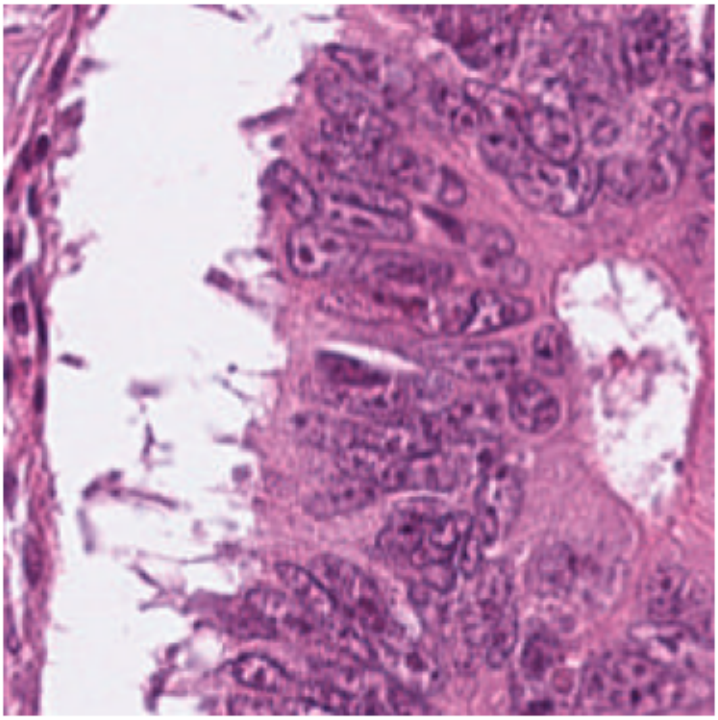
```
    zf.extractall(".")
print("completed.")

# 确认类别
classes = sorted([d.name for d in data_dir.iterdir() if d.is_dir()])
print(f"Pathology Classes ({len(classes)}): {classes}")
```

```
completed.
Pathology Classes (9): ['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM',
'STR', 'TUM']
```

数统与校验

[4]:
```
# 图像 patch 总数
total_images = sum([len(files) for r, d, files in os.walk(data_dir)])
print(total_images)

# 查看图像 patch 规格
sample_path = next(data_dir.glob("**/*.tif"))
with Image.open(sample_path) as img:
    print(f"格式: {img.format} | 尺寸: {img.size}")
    plt.imshow(img)
    plt.axis('off')
    plt.show()
```

```
100000
格式: TIFF | 尺寸: (224, 224)
```



0.1.4 Step 2 数据预处理与划分

[5]:
```
# 数据增强
data_transforms = {
    'train': transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomVerticalFlip(p=0.5),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
```

```
# 读取完整数据集
full_dataset = datasets.ImageFolder(root=data_dir)

# 划分训练集与验证集 (80% 训练, 20% 验证)
train_size = int(0.8 * len(full_dataset))
val_size = len(full_dataset) - train_size
train_set, val_set = random_split(full_dataset, [train_size, val_size])

# 包装数据集以应用变换
class WrapTransform(torch.utils.data.Dataset):
    def __init__(self, subset, transform=None):
        self.subset = subset
        self.transform = transform
    def __getitem__(self, index):
        x, y = self.subset[index]
        if self.transform: x = self.transform(x)
        return x, y
    def __len__(self):
        return len(self.subset)

train_data = WrapTransform(train_set, transform=data_transforms['train'])
val_data = WrapTransform(val_set, transform=data_transforms['val'])

# 创建 DataLoader
dataloaders = {
    'train': DataLoader(train_data, batch_size=128, shuffle=True,␣
 ↪num_workers=4, pin_memory=True),
    'val': DataLoader(val_data, batch_size=128, shuffle=False, num_workers=4,␣
 ↪pin_memory=True)
}

print(f"训练集: {len(train_data)} | 验证集: {len(val_data)}")
```

训练集: 80000 | 验证集: 20000

数据预处理是深度学习的关键环节：1. **标准化**：采用 ImageNet 的均值和方差，使输入数据分布与

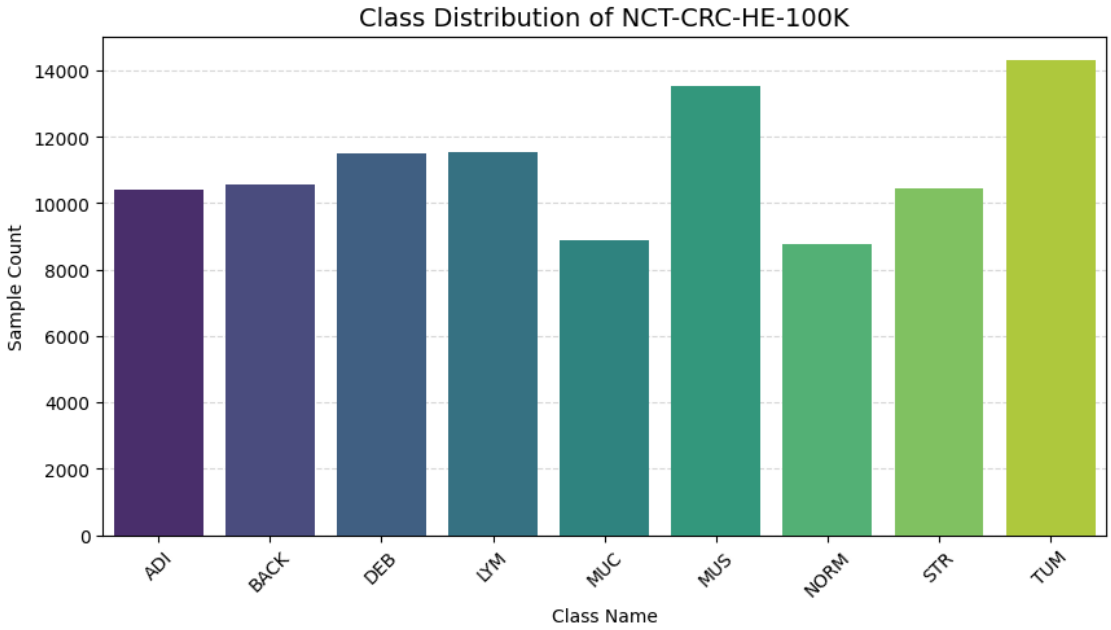
预训练模型相匹配，加速收敛。2. **数据划分**：数据集按 8:2 分割为训练集和验证集，防止模型评估时的“数据泄露”。3. **数据增强**：引入随机水平和垂直翻转，提升模型的泛化能力，防止过拟合。

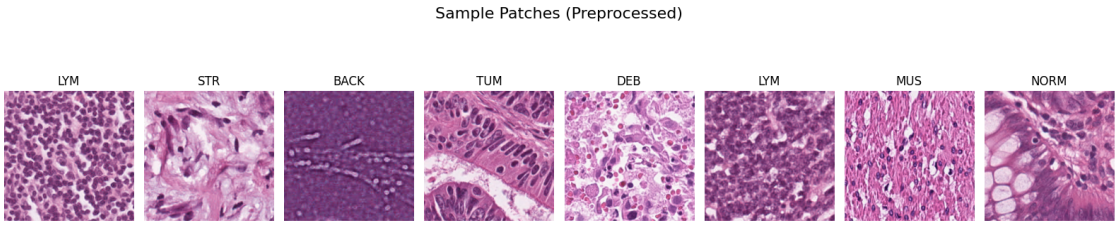
0.1.5 Step 3 统计分析

[6]:
```
targets = [s[1] for s in full_dataset.samples]
class_indices = np.unique(targets)
class_counts = np.bincount(targets)
class_names = full_dataset.classes

# 1. 统计表格
df_stats = pd.DataFrame({
    'Class Name': class_names,
    'Sample Count': class_counts,
    'Percentage': [f"{c/len(full_dataset)*100:.2f}%" for c in class_counts]
})
display(df_stats)

# 2. 类别分布直方图
plt.figure(figsize=(10, 5))
plt.grid(axis='y', linestyle='--', alpha=0.5, zorder=0)
sns.barplot(x='Class Name', y='Sample Count', hue='Class Name', data=df_stats,␣
 ↪palette="viridis", legend=False, zorder=3)
plt.title("Class Distribution of NCT-CRC-HE-100K", fontsize=14)
plt.xticks(rotation=45)
plt.show()

# 3. 样本展示
def imshow(inp, title=None):
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title: plt.title(title)
    plt.axis('off')
```

```
inputs, classes_idx = next(iter(dataloaders['val']))

plt.figure(figsize=(16, 4))
for i in range(8):
    plt.subplot(1, 8, i+1)
    imshow(inputs[i], title=class_names[classes_idx[i]])
plt.suptitle("Sample Patches (Preprocessed)", fontsize=16)
plt.tight_layout()
plt.show()
```

	Class Name	Sample Count	Percentage
0	ADI	10407	10.41%
1	BACK	10566	10.57%
2	DEB	11512	11.51%
3	LYM	11557	11.56%
4	MUC	8896	8.90%
5	MUS	13536	13.54%
6	NORM	8763	8.76%
7	STR	10446	10.45%
8	TUM	14317	14.32%



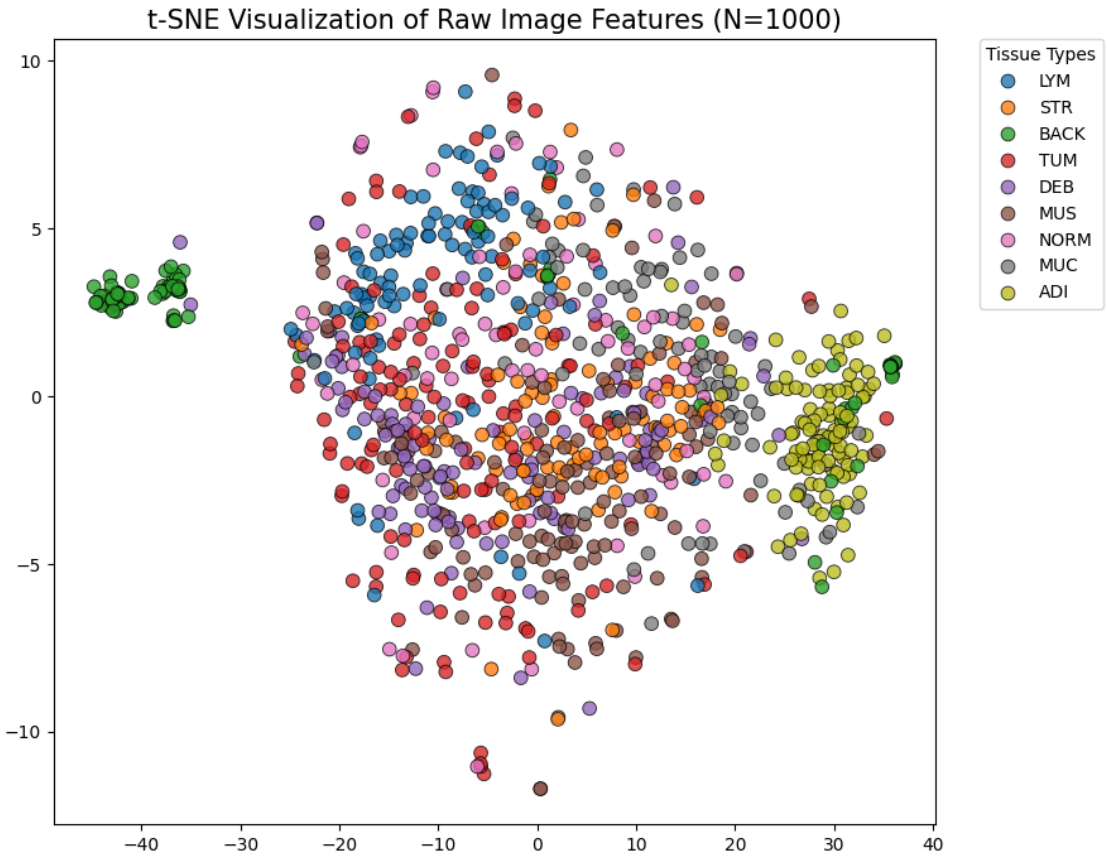


类别分布：从柱状图可以看出，各组织类别的样本量分布相对均衡，这意味着我们在训练时不需要过多的类别加权处理。

可视化 (t-SNE & PCA)

[7]:
```
# 1. 抽取样本
extracted_samples = 1000
features_list = []
labels_list = []
current_count = 0


for inputs_batch, labels_batch in dataloaders['val']:
    if current_count >= extracted_samples:
        break

    # 降采样图片以减少计算量 (224x224 -> 32x32)
    small_imgs = F.interpolate(inputs_batch, size=(32, 32), mode='bilinear',␣
 ↪align_corners=False)

    # 展平 [B, 3, 32, 32] -> [B, 3072]
    flat = small_imgs.view(inputs_batch.size(0), -1)

    features_list.append(flat.numpy())
    labels_list.extend(labels_batch.numpy())
    current_count += inputs_batch.size(0)

X = np.concatenate(features_list, axis=0)[:extracted_samples]
```

```
y = np.array(labels_list)[:extracted_samples]

# 2. PCA 降维
pca = PCA(n_components=50, random_state=2026)
X_pca = pca.fit_transform(X)

# 3. t-SNE 降维
tsne = TSNE(n_components=2, perplexity=30, random_state=2026, init='pca',␣
 ↪learning_rate='auto')
X_embedded = tsne.fit_transform(X_pca)

# 4. 绘图
plt.figure(figsize=(9, 7))

hue_labels = [class_names[i] for i in y]

scatter = sns.scatterplot(x=X_embedded[:,0], y=X_embedded[:,1], hue=hue_labels,
                          palette="tab10", s=60, alpha=0.8, edgecolor='k')
plt.title(f"t-SNE Visualization of Raw Image Features (N={extracted_samples})",␣
 ↪fontsize=15)
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0., title="Tissue␣
 ↪Types")
plt.tight_layout()
plt.show()
```



t-SNE：该图展示了原始像素特征在二维空间的分布，可以看到不同颜色的点群出现自然聚类（例如 ADI 脂肪组织与 BACK 背景自成一派），说明数据在未经训练的情况下已具有良好的可区分性。

0.1.6 Step 4 预测模型建立

模型构建

[8]:
```
if 'full_dataset' in locals():
    class_names = full_dataset.classes
else:
    class_names = sorted([d.name for d in Path("./NCT-CRC-HE-100K").iterdir()␣
 ↪if d.is_dir()])

# 模型构建
model = models.resnet50(weights=models.ResNet50_Weights.DEFAULT)
```

```
# 修改全连接层以匹配类别数量
feature_dim = model.fc.in_features
model.fc = nn.Linear(feature_dim, len(class_names))

model = model.to(device)

# 定义损失函数
criterion = nn.CrossEntropyLoss()

# 优化器
optimizer = optim.AdamW(model.parameters(), lr=1e-4, weight_decay=1e-3)

# 混合精度训练 Scalar
scaler = torch.amp.GradScaler('cuda')
```

选择了 **ResNet50** 作为骨干网络 * **迁移学习策略**：加载了在 ImageNet 上预训练的权重 * **网络改造**：将全连接层 (FC) 的输出维度修改为 9，以匹配本任务的类别数

模型训练与验证

[9]:
```
num_epochs = 3
history = {'train_loss': [], 'train_acc': [], 'val_loss': [], 'val_acc': []}

for epoch in range(num_epochs):
    print(f'Epoch {epoch+1}/{num_epochs}')

    for phase in ['train', 'val']:
        if phase == 'train':
            model.train()
        else:
            model.eval()

        running_loss = 0.0
        running_corrects = 0

        loop = tqdm(dataloaders[phase], file=sys.stdout, ncols=80)
```

```
        for inputs, labels in loop:
            inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()

            with torch.set_grad_enabled(phase == 'train'):
                with torch.amp.autocast('cuda'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                if phase == 'train':
                    scaler.scale(loss).backward()
                    scaler.step(optimizer)
                    scaler.update()

            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)

            loop.set_postfix(loss=loss.item())

        epoch_loss = running_loss / len(dataloaders[phase].dataset)
        epoch_acc = running_corrects.double() / len(dataloaders[phase].dataset)

        history[f'{phase}_loss'].append(epoch_loss)
        history[f'{phase}_acc'].append(epoch_acc.item())

        print(f'{phase.title()} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')

print('Completed!')
```

```
Epoch 1/3
100%|               | 625/625 [01:04<00:00,  9.68it/s, loss=0.0455]
Train Loss: 0.1216 Acc: 0.9683
100%|               | 157/157 [00:07<00:00, 21.32it/s, loss=0.00843]
Val Loss: 0.0236 Acc: 0.9928
```

```
Epoch 2/3
100%|                | 625/625 [01:03<00:00,  9.79it/s, loss=0.037]
Train Loss: 0.0186 Acc: 0.9942
100%|               | 157/157 [00:07<00:00, 21.48it/s, loss=0.00181]
Val Loss: 0.0122 Acc: 0.9961
Epoch 3/3
100%|                | 625/625 [01:03<00:00,  9.84it/s, loss=0.0875]
Train Loss: 0.0112 Acc: 0.9964
100%|               | 157/157 [00:07<00:00, 21.09it/s, loss=0.00463]
Val Loss: 0.0102 Acc: 0.9972
Completed!
```

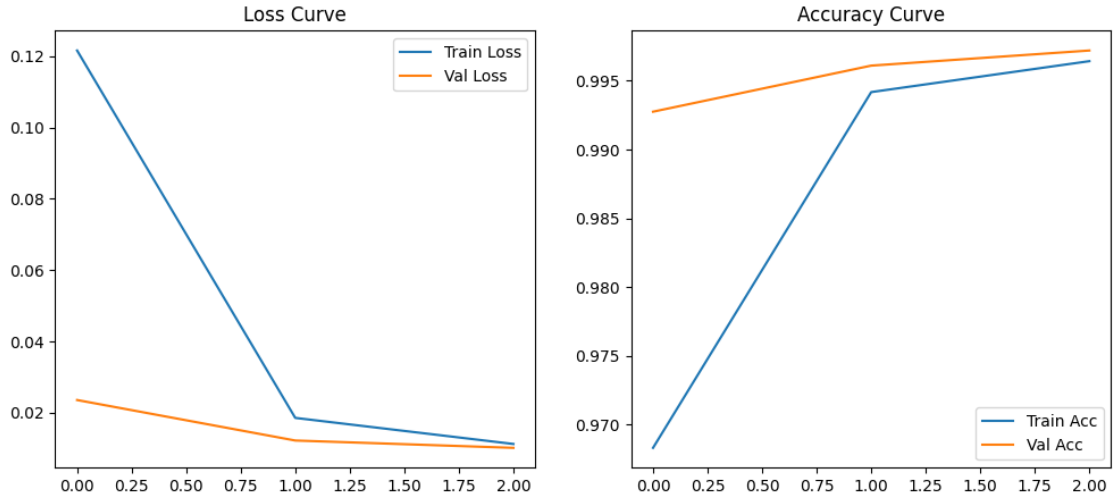
模型收敛非常好，在 3 个 Epoch 内，模型在验证集上的准确率已达到非常高水平

0.1.7 Step 5 预测模型评估与可视化

[16]:
```python
# 1. 绘制训练曲线
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history['train_loss'], label='Train Loss')
plt.plot(history['val_loss'], label='Val Loss')
plt.title("Loss Curve")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history['train_acc'], label='Train Acc')
plt.plot(history['val_acc'], label='Val Acc')
plt.title("Accuracy Curve")
plt.legend()
plt.show()

# 2. 计算指标
y_true = []
y_pred = []
y_probs = []

model.eval()
with torch.no_grad():
```

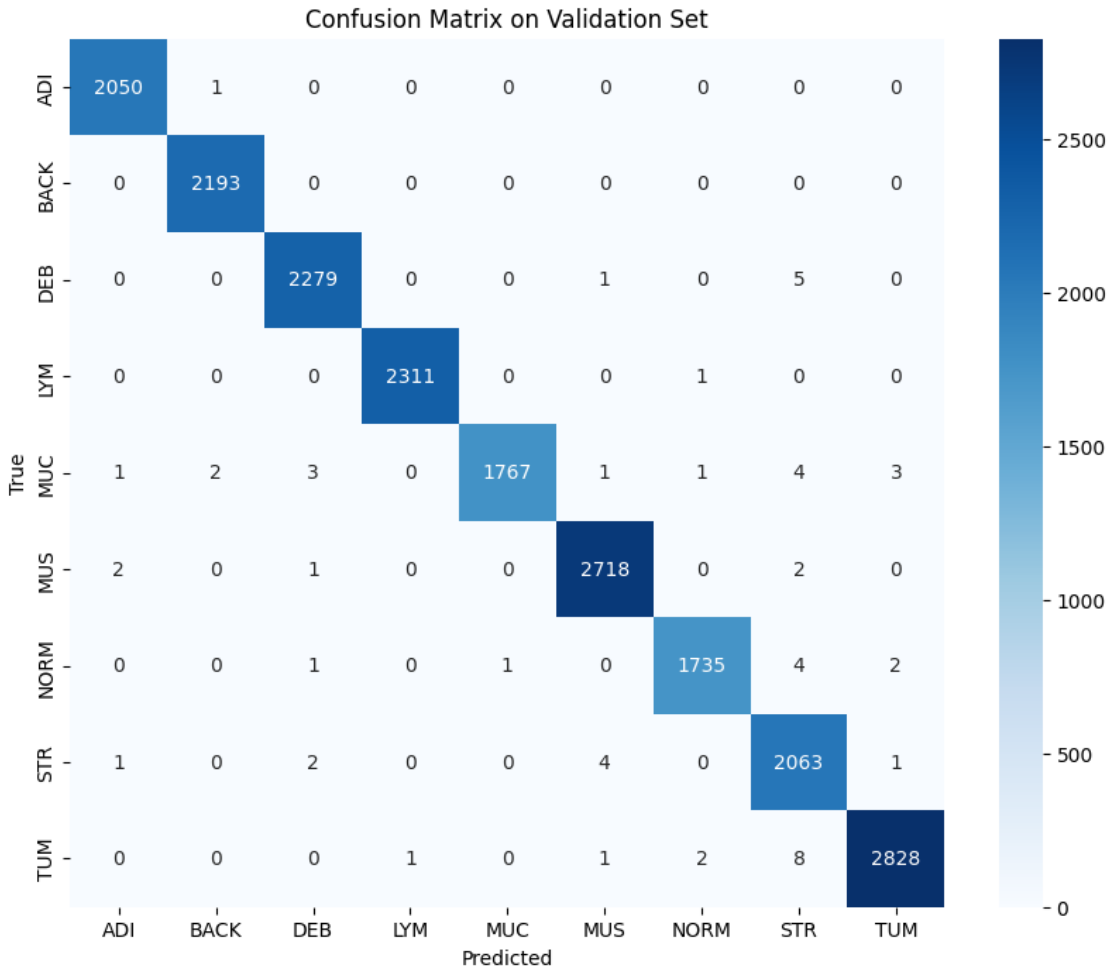
```
    for inputs, labels in tqdm(dataloaders['val']):
        inputs = inputs.to(device)
        outputs = model(inputs)

        # 获取类别预测
        _, preds = torch.max(outputs, 1)
        # 获取概率预测
        probs = torch.softmax(outputs, dim=1)

        y_true.extend(labels.cpu().numpy())
        y_pred.extend(preds.cpu().numpy())
        y_probs.extend(probs.cpu().numpy())

y_probs = np.array(y_probs)

# 精确率、召回率、F1
print(classification_report(y_true, y_pred, target_names=class_names))

# 绘制混淆矩阵
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix on Validation Set')
plt.show()

# 将标签二值化
y_test_bin = label_binarize(y_true, classes=range(len(class_names)))
n_classes = y_test_bin.shape[1]

# 计算每一类的 ROC 曲线和 AUC
fpr = dict()
tpr = dict()
roc_auc = dict()
```

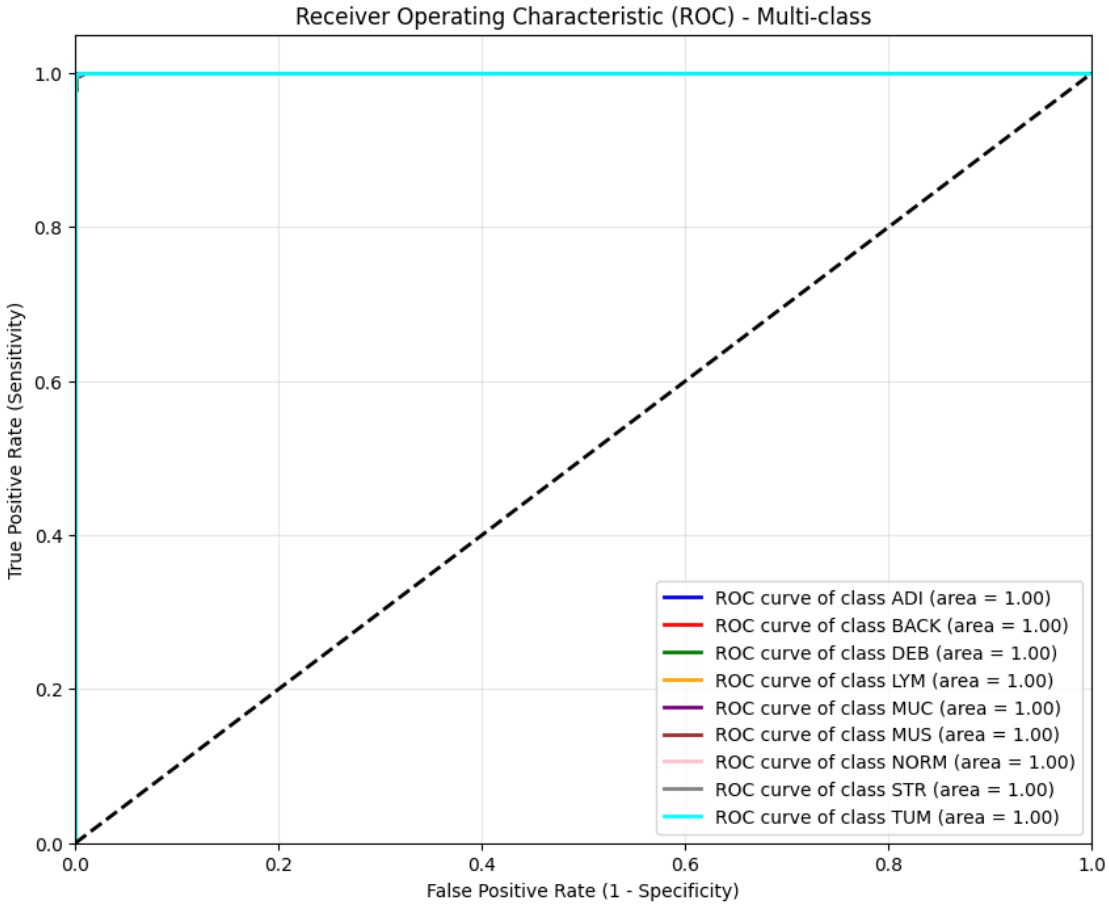
```
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_probs[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# 绘图
plt.figure(figsize=(10, 8))
colors = cycle(['blue', 'red', 'green', 'orange', 'purple', 'brown', 'pink',␣
 ↪'gray', 'cyan'])

for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(class_names[i], roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.title('Receiver Operating Characteristic (ROC) - Multi-class')
plt.legend(loc="lower right")
plt.grid(alpha=0.3)
plt.show()
```



```
100%|      | 157/157 [00:10<00:00, 15.44it/s]
```

	precision	recall	f1-score	support
ADI	1.00	1.00	1.00	2051
BACK	1.00	1.00	1.00	2193
DEB	1.00	1.00	1.00	2285
LYM	1.00	1.00	1.00	2312
MUC	1.00	0.99	1.00	1782
MUS	1.00	1.00	1.00	2723
NORM	1.00	1.00	1.00	1743
STR	0.99	1.00	0.99	2071
TUM	1.00	1.00	1.00	2840
accuracy			1.00	20000
macro avg	1.00	1.00	1.00	20000
weighted avg	1.00	1.00	1.00	20000


Confusion Matrix on Validation Set
True
Predicted
ADI
BACK
DEB
LYM
MUC
MUS
NORM
STR
TUM
2050 1 0 0 0 0 0 0 0
0 2193 0 0 0 0 0 0 0
0 0 2279 0 0 1 0 5 0
0 0 0 2311 0 0 1 0 0
1 2 3 0 1767 1 1 4 3
2 0 1 0 0 2718 0 2 0
0 0 1 0 1 0 1735 4 2
1 0 2 0 0 4 0 2063 1
0 0 0 1 0 1 2 8 2828
2500
2000
1500
1000
500
0



1. 定量评估

- **混淆矩阵 (Confusion Matrix)**：对角线颜色深，说明大部分样本被正确分类。模型的主要混淆可能发生在 **STR (基质)** 和 **MUS (平滑肌)** 之间，这在病理学上是合理的，因为两者在 H&E 染色下形态较为接
- **分类报告**：各项指标 (Precision, Recall, F1-score) 均表现优异
- **ROC 曲线**：该数据集分类准确性极好，这里主要是演示流程，ROC 曲线是评估模型分类效能的重要指标

[14]:
```
class GradCAM:
    def __init__(self, model, target_layer):
        self.model = model
        self.target_layer = target_layer
        self.gradients = None
```

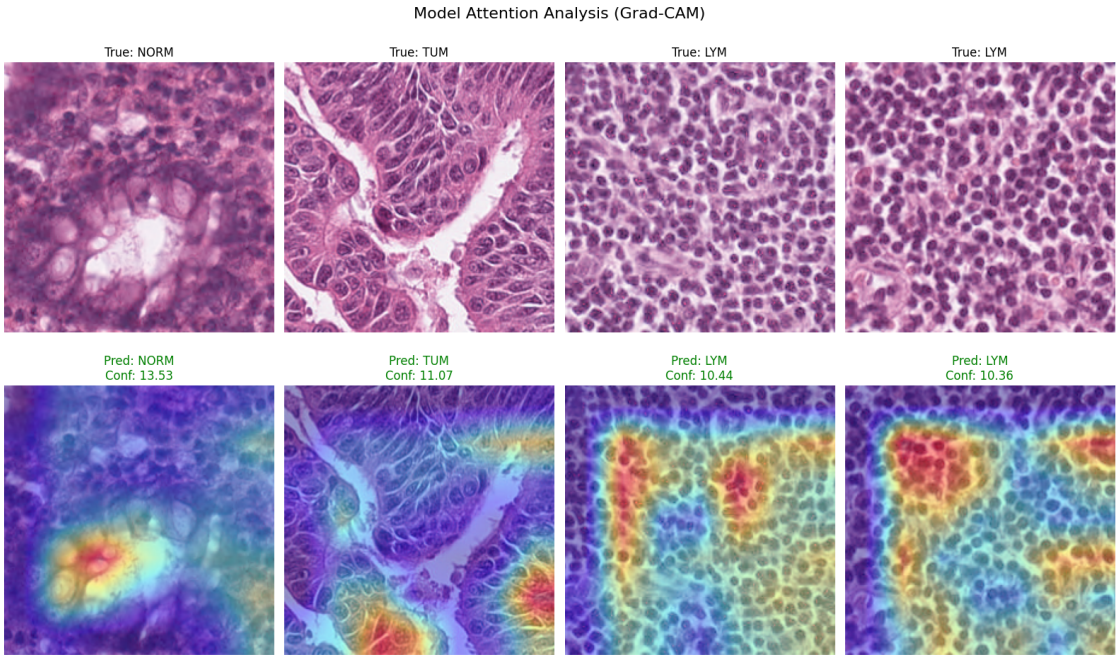
```
        self.activations = None

        self.target_layer.register_forward_hook(self.save_activation)
        self.target_layer.register_full_backward_hook(self.save_gradient)

    def save_activation(self, module, input, output):
        self.activations = output

    def save_gradient(self, module, grad_input, grad_output):
        self.gradients = grad_output[0]

    def generate_cam(self, input_image, target_class=None):

        model.eval()
        output = self.model(input_image)

        if target_class is None:
            target_class = output.argmax(dim=1).item()



        self.model.zero_grad()
        score = output[0, target_class]
        score.backward()

        gradients = self.gradients[0] # [C, H, W]
        activations = self.activations[0] # [C, H, W]

        weights = torch.mean(gradients, dim=(1, 2), keepdim=True)
        cam = torch.sum(weights * activations, dim=0)
        cam = F.relu(cam)

        cam = cam - cam.min()
        cam = cam / (cam.max() + 1e-7)
        return cam.detach().cpu().numpy(), target_class, output.max().item()
```

```
# ResNet50 的结构：layer4 是最后一个残差块，[-1] 是该块的最后一层
target_layer = model.layer4[-1]
grad_cam = GradCAM(model, target_layer)

# 随机抽取 4 张验证集图片
demo_loader = torch.utils.data.DataLoader(val_data, batch_size=1, shuffle=True)
demo_images = []
demo_labels = []

for img, label in demo_loader:
    demo_images.append(img)
    demo_labels.append(label)
    if len(demo_images) == 4:
        break

plt.figure(figsize=(16, 10))

inv_normalize = transforms.Normalize(
    mean=[-0.485/0.229, -0.456/0.224, -0.406/0.225],
    std=[1/0.229, 1/0.224, 1/0.225]
)


for i in range(4):
    input_tensor = demo_images[i].to(device)
    input_tensor.requires_grad = True
    label_idx = demo_labels[i].item()

    # 生成热图
    mask, pred_class, conf = grad_cam.generate_cam(input_tensor)

    # 图像处理用于显示
    orig_img = inv_normalize(input_tensor[0]).cpu().detach().permute(1, 2, 0).
 ↪numpy()
    orig_img = np.clip(orig_img, 0, 1)
```

```
    # 将 Heatmap 缩放到原图大小
    heatmap = Image.fromarray(np.uint8(255 * mask))
    heatmap = heatmap.resize((224, 224), resample=Image.BILINEAR)
    heatmap = np.array(heatmap) / 255.0

    # 叠加显示
    heatmap_colored = plt.get_cmap('jet')(heatmap)[:, :, :3] # 取 RGB
    overlay = 0.6 * orig_img + 0.4 * heatmap_colored



    plt.subplot(2, 4, i + 1)
    plt.imshow(orig_img)
    plt.title(f"True: {class_names[label_idx]}", fontsize=12)
    plt.axis('off')



    plt.subplot(2, 4, i + 5)
    plt.imshow(overlay)
    pred_name = class_names[pred_class]
    color = 'green' if pred_class == label_idx else 'red'
    plt.title(f"Pred: {pred_name}\nConf: {conf:.2f}", fontsize=12, color=color)
    plt.axis('off')

plt.suptitle("Model Attention Analysis (Grad-CAM)", fontsize=16)
plt.tight_layout()
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range
[0.051764730364084244..1.0000000238418578].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range
[0.05411766842007637..1.0000000238418578].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range
[0.0564705990254879..1.0000000238418578].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

```
floats or [0..255] for integers). Got range
[0.05411766842007637..1.0000000238418578].
```



2. 定性评估 (可解释性分析)

- **Grad-CAM 热图**：可视化结果显示，红色高亮区域表示模型在分类时最关注的区域。如果红色区域集中在细胞核密集处（对于 LYM/TUM）或特定的纹理结构（对于 MUS/STR），说明模型学到了正确的病理形态学特征，而非学习了背景噪声。
  - 在 **TUM (肿瘤)** 样本中，热图高亮区域集中在异型性明显的上皮细胞核；
  - 在 **LYM (淋巴细胞)** 样本中，模型关注到了由于细胞核密集而显得深染的淋巴细胞核簇